COMPARING CONVENTIONAL EXECUTABLE SEMANTICS

Grigore Rosu

CS422 – Programming Language Design

Overview

- We next discuss the conventional executable semantics approaches in depth, aiming to understand their pros and cons
- Our approach is to extend each semantics of IMP with several common features (we call the resulting language IMP++)
  - *Variable increment* – this will add side effects to expressions
  - *Output* – this will require changes in the configuration
  - *Abrupt termination* – this requires explicit handling of control
  - *Dynamic threads* – this requires handling concurrency and sharing
  - *Local variables* – this requires handling environments
- Important note!
  - To keep this language design experiment realistic, we will pretend that each extension of IMP is final, i.e., we do not pro-actively take semantic decisions when defining a feature that will help the definition of other features that we know will be added later on.

IMP++: Variable Increment

- Syntax:

$$AExp \quad ::= \quad \ldots \quad | \quad {+}{+}\, Id$$

- Variable increment is very common (C, C++, Java, etc.)
  - We only consider post-increment
- The problem with increment in some semantic approaches is that it adds side effects to expressions. Therefore, if one did not pro-actively account for that then one needs to change many existing and unrelated semantics rules, if not all.

IMP++: Variable Increment Big-Step SOS

- Previous big-step SOS rules had the form:

$$\frac{\langle a_1, \sigma\rangle \Downarrow \langle i_1\rangle \qquad \langle a_2, \sigma\rangle \Downarrow \langle i_2\rangle}{\langle a_1 \ / \ a_2, \sigma\rangle \Downarrow \langle i_1 \ /_{Int} \ i_2\rangle} \text{, where } i_2 \neq 0$$

- Big-step SOS is the most affected by side effects
  - Needs to change its sequents from $\langle a, \sigma\rangle \Downarrow \langle i\rangle$ to $\langle a, \sigma\rangle \Downarrow \langle i, \sigma'\rangle$
  - And all the existing rules accordingly, e.g.:

$$\frac{\langle a_1, \sigma\rangle \Downarrow \langle i_1, \sigma_1\rangle, \ \langle a_2, \sigma_1\rangle \Downarrow \langle i_2, \sigma_2\rangle}{\langle a_1 \ / \ a_2, \sigma\rangle \Downarrow \langle i_1 /_{Int} i_2, \sigma_2\rangle} \quad \text{where } i_2 \neq 0$$

IMP++: Variable Increment Big-Step SOS

- Recall IMP operators like / were non-deterministically strict. Here is a desperate attempt to achieve that with big-step SOS

$$\frac{\langle a_1, \sigma\rangle \Downarrow \langle i_1, \sigma_1\rangle,\ \langle a_2, \sigma_1\rangle \Downarrow \langle i_2, \sigma_2\rangle}{\langle a_1\ /\ a_2, \sigma\rangle \Downarrow \langle i_1 /_{Int} i_2, \sigma_2\rangle}\ ,\quad \text{where } i_2 \neq 0$$

$$\frac{\langle a_1, \sigma_2\rangle \Downarrow \langle i_1, \sigma_1\rangle,\ \langle a_2, \sigma\rangle \Downarrow \langle i_2, \sigma_2\rangle}{\langle a_1\ /\ a_2, \sigma\rangle \Downarrow \langle i_1 /_{Int} i_2, \sigma_1\rangle}\ ,\quad \text{where } i_2 \neq 0$$

  - All we got is "non-deterministic choice" strictness: choose an order, then evaluate the arguments in that order
    - Some behaviors are therefore lost
    - This is relatively acceptable in many cases (loss of behaviors when we add threads is going to be much worse)

IMP++: Variable Increment Big-Step SOS

- We are now ready to add the big-step SOS for variable increment (this is easy now, the hard part was to get here):

$$\langle {+}{+}\, x, \sigma \rangle \Downarrow \langle \sigma(x) +_{Int} 1, \sigma[(\sigma(x) +_{Int} 1)/x] \rangle$$

- Example:
  - How many values can the following expression possibly evaluate to under the big-step SOS of IMP++ above (assume x is initially 1)?

$$({+}{+}\, x) / ({+}{+}\, x / x)$$

  - Can it evaluate to 0 or even be underfined under a fully non-deterministic evaluation strategy?

IMP++: Variable Increment Small-Step SOS

- Previous small-step SOS rules had the form:

$$\frac{\langle a_1, \sigma \rangle \to \langle a_1', \sigma \rangle}{\langle a_1 \ / \ a_2, \sigma \rangle \to \langle a_1' \ / \ a_2, \sigma \rangle}$$

- Small-step SOS less affected than big-step SOS, but still requires many rule changes to account for the side effects:

$$\frac{\langle a_1, \sigma \rangle \to \langle a_1', \sigma_1 \rangle}{\langle a_1 \ / \ a_2, \sigma \rangle \to \langle a_1' \ / \ a_2, \sigma_1 \rangle}$$

IMP++: Variable Increment Small-Step SOS

- Since small-step SOS "gets back to the top" at each step, it actually does not lose any non-deterministic behaviors
  - We get fully non-deterministic evaluation strategies for all the IMP constructs instead of "non-deterministic choice" ones
- The semantics of variable increment almost the same as in big-step SOS:

$$\langle {+}{+}\, x, \sigma \rangle \rightarrow \langle \sigma(x) +_{Int} 1, \sigma[(\sigma(x) +_{Int} 1)/x] \rangle$$

IMP++: Variable Increment MSOS

- Previous MSOS rules had the form:

$$\frac{a_1 \to a_1'}{a_1 \,/\, a_2 \to a_1' \,/\, a_2}$$

- All semantic changes are hidden within labels, which are implicitly propagated through the general MSOS mechanism
- Consequently, the MSOS of IMP only needs the following rule to accommodate variable updates; nothing else changes!

$$\texttt{++}\, x \xrightarrow{\{\text{state}=\sigma,\text{state}'=\sigma[(\sigma(x)+_{Int}1)/x],\ldots\}} \sigma(x) +_{Int} 1$$

IMP++: Variable Increment Reduction Semantics with Eval. Contexts

- Previous RSEC evaluation contexts and rules had the form:

$$\begin{array}{rcl} Context & ::= & \square \\ & | & Context + AExp \mid AExp + Context \end{array}$$

$$i_1 / i_2 \rightarrow i_1 /_{Int} i_2, \quad \text{when } i_2 \neq 0$$

$$\langle c, \sigma \rangle [x] \rightarrow \langle c, \sigma \rangle [\sigma(x)]$$

- Evaluation contexts, together with the characteristic rule of RSEC, allows for compact unconditional rules, mentioning only what is needed from the entire configuration
- Consequently, the RSED of IMP only needs the following rule to accommodate variable updates; nothing else changes!

$$\langle c, \sigma \rangle [{+}{+}\, x] \rightarrow \langle c, \sigma[(\sigma(x) +_{Int} 1)/x] \rangle [\sigma(x) +_{Int} 1]$$

IMP++: Variable Increment CHAM

- Previous CHAM heating/cooling/reaction rules had the form:

$$
\begin{aligned}
a_1 \,/\, a_2 \rightsquigarrow c &\rightleftharpoons a_1 \rightsquigarrow \square \,/\, a_2 \rightsquigarrow c \\
a_1 \,/\, a_2 \rightsquigarrow c &\rightleftharpoons a_2 \rightsquigarrow a_1 \,/\, \square \rightsquigarrow c \\
i_1 \,/\, i_2 \rightsquigarrow c &\rightarrow i_1 /_{Int} i_2 \rightsquigarrow c \quad \text{when } i_2 \neq 0 \\
\{\!|x \rightsquigarrow c|\!\}\ \{\!|x \mapsto i \triangleright \sigma|\!\} &\rightarrow \{\!|i \rightsquigarrow c|\!\}\ \{\!|x \mapsto i \triangleright \sigma|\!\}
\end{aligned}
$$

- Since the heating/cooling rules achieve the role of the evaluation contexts and since one can only mention the necessary molecules in each rule, one does not need to change anything either!
- All one needs to do is to add the following rule:

$$\{\!|{+}{+}x \rightsquigarrow c|\!\}\ \{\!|x \mapsto i \triangleright \sigma|\!\} \rightarrow \{\!|i +_{Int} 1 \rightsquigarrow c|\!\}, \{\!|x \mapsto i +_{Int} 1 \triangleright \sigma|\!\}$$